

2º curso / 2º cuatr.
Grado Ing. Inform.
Doble Grado Ing.
Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos): Alberto Jesús Durán López

Grupo de prácticas:1

Fecha de entrega:

Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

CÓDIGO FUENTE: `if-clauseModificado.c`

```
#include <stdlib.h>
#include <omp.h>

int main(int argc, char **argv)
{
    int i, n=20, tid;
    int a[n], suma=0, sumalocal;

    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }
    int x= atoi(argv[2]);
    n = atoi(argv[1]);

    if (n>20)
        n=20;

    for (i=0; i<n; i++) {
        a[i] = i;
    }

    #pragma omp parallel num_threads(x) if(n>4) default(none) \
    private(sumalocal,tid) shared(a,suma,n)
    {
        sumalocal=0;
        tid= omp_get_thread_num();

        #pragma omp for private(i) schedule(static) nowait
        for (i=0; i<n; i++){
            sumalocal += a[i];
        }
    }
```

```

        printf(" thread %d suma de a[%d]=%d sumalocal=%d \n",
               tid,i,a[i],sumalocal);
    }

    #pragma omp atomic
    suma += sumalocal;
    #pragma omp barrier

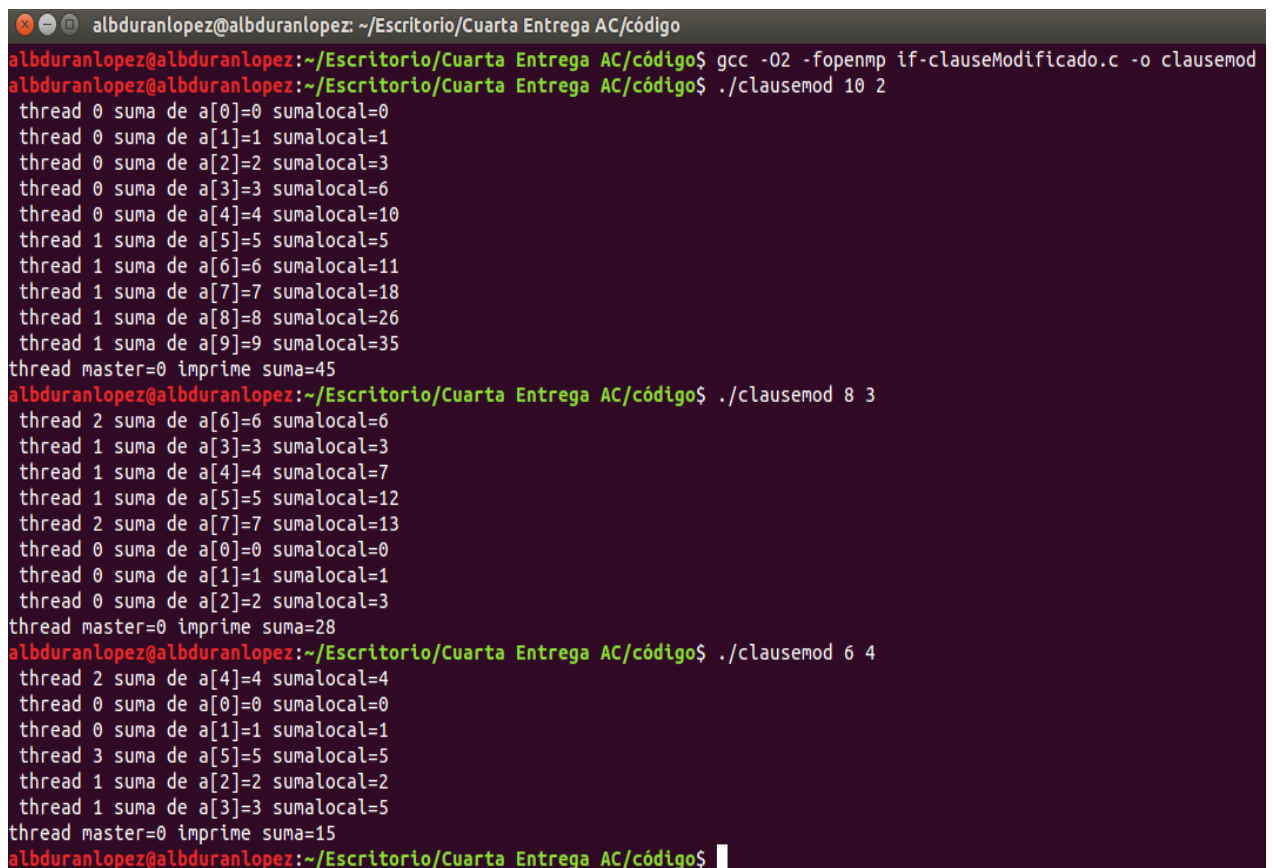
    #pragma omp master
    printf("thread master=%d imprime suma=%d\n",tid,suma);
}
}

```

RESPUESTA:

Justo después de la clausula “num_threads(x)” se indica que solo se ejecuta el código en paralelo cuando n sea mayor que 4 ; if(n>4)

Cuando hay más de 4 interacciones, se crearan las x hebras que hemos introducido como segundo argumento.

CAPTURAS DE PANTALLA:


```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -O2 -fopenmp if-clauseModificado.c -o clausemod
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./clausemod 10 2
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread 0 suma de a[3]=3 sumalocal=6
thread 0 suma de a[4]=4 sumalocal=10
thread 1 suma de a[5]=5 sumalocal=5
thread 1 suma de a[6]=6 sumalocal=11
thread 1 suma de a[7]=7 sumalocal=18
thread 1 suma de a[8]=8 sumalocal=26
thread 1 suma de a[9]=9 sumalocal=35
thread master=0 imprime suma=45
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./clausemod 8 3
thread 2 suma de a[6]=6 sumalocal=6
thread 1 suma de a[3]=3 sumalocal=3
thread 1 suma de a[4]=4 sumalocal=7
thread 1 suma de a[5]=5 sumalocal=12
thread 2 suma de a[7]=7 sumalocal=13
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread master=0 imprime suma=28
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./clausemod 6 4
thread 2 suma de a[4]=4 sumalocal=4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 3 suma de a[5]=5 sumalocal=5
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread master=0 imprime suma=15
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$

```

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,...15)
- chunk= 1, 2 y 4

Tabla 1 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-clause.d.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	0	1	0	0	0
7	1	1	1	0	0	1	0	0	0
8	0	0	0	0	0	0	1	1	1
9	1	0	0	0	0	0	1	1	1
10	0	1	0	0	0	0	1	1	1
11	1	1	0	0	0	0	1	1	1
12	0	0	1	0	0	0	0	0	0
13	1	0	1	0	0	0	0	0	0
14	0	1	1	0	0	0	0	0	1
15	1	1	1	0	0	0	0	0	1

(**export OMP_NUM_THREADS = 2**)

Para rellenar la tabla, cambiamos el valor de n(que son las iteraciones del programa) a 16 y cambiamos el valor del chunk que se lo pasamos como parámetro. Así para los tres programas. Nos queda por ejemplo:

```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp
-O2 schedule-clause.c -o clause
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./clause 1
thread 0 suma a[0] suma=0
thread 0 suma a[2] suma=2
thread 0 suma a[4] suma=6
thread 0 suma a[6] suma=12
thread 0 suma a[8] suma=20
thread 0 suma a[10] suma=30
thread 0 suma a[12] suma=42
thread 0 suma a[14] suma=56
thread 1 suma a[1] suma=1
thread 1 suma a[3] suma=4
thread 1 suma a[5] suma=9
thread 1 suma a[7] suma=16
thread 1 suma a[9] suma=25
thread 1 suma a[11] suma=36
thread 1 suma a[13] suma=49
thread 1 suma a[15] suma=64
Fuera de 'parallel for' suma=64
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./clause 2
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[4] suma=5
thread 0 suma a[5] suma=10
thread 0 suma a[8] suma=18
thread 0 suma a[9] suma=27
thread 0 suma a[12] suma=39
thread 0 suma a[13] suma=52
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 1 suma a[6] suma=11
thread 1 suma a[7] suma=18
thread 1 suma a[10] suma=28
thread 1 suma a[11] suma=39
thread 1 suma a[14] suma=53
thread 1 suma a[15] suma=68
Fuera de 'parallel for' suma=68
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./clause 4
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[3] suma=6
thread 0 suma a[8] suma=14
thread 0 suma a[9] suma=23
thread 0 suma a[10] suma=33
thread 0 suma a[11] suma=44
thread 1 suma a[4] suma=4
thread 1 suma a[5] suma=9
thread 1 suma a[6] suma=15
thread 1 suma a[7] suma=22
thread 1 suma a[12] suma=34
thread 1 suma a[13] suma=47
thread 1 suma a[14] suma=61
thread 1 suma a[15] suma=76
Fuera de 'parallel for' suma=76

```

(b) Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

(export OMP_NUM_THREADS=4)

Tabla 2 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule- clause.c			schedule- claused.c			schedule- clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	3	0	3	0	3	3
1	1	0	0	0	0	3	0	3	3
2	2	1	0	2	1	3	0	3	3
3	3	1	0	1	1	3	0	3	3
4	0	2	1	3	3	2	1	2	2
5	1	2	1	3	3	2	1	2	2
6	2	3	1	3	2	2	1	2	2
7	3	3	1	3	2	2	3	0	2
8	0	0	2	3	0	0	3	0	1
9	1	0	2	3	0	0	3	0	1
10	2	1	2	3	0	0	2	1	1
11	3	1	2	3	0	0	2	1	1
12	0	2	3	3	0	1	0	0	0
13	1	2	3	3	0	1	0	0	0
14	2	3	3	3	0	1	0	0	0
15	3	3	3	3	0	1	0	0	0

Escriba en el cuaderno de prácticas las diferencias en el comportamiento de `schedule()` con `static`, `dynamic` y `guided`.

RESPUESTA:

Static: Distribuye un subconjunto de iteraciones en cada thread de modo circular. El tamaño viene dado por “chunk”. Si no se especifica el chunk, la distribución es por bloques.

Dynamic: Es parecido a static, pero los bloques se reparten dinámicamente a medida que los hilos van finalizando. Su valor por defecto es 1.

Guided: Cada thread coge dinámicamente bloques de iteraciones. El bloque inicialmente es grande (nº iteraciones sin asignar / nº threads) y va bajando hasta tamaño “chunk”

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char **argv)
{
    int i, n=16, chunk, a[n], suma=0;
    int modifier;
    omp_sched_t kind;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    //dyn-var = omp_get_dynamic();
    //nthreads-var = omp_get_max_threads();
    //thread-limit-var=omp_get_thread_limit();
    //run-sched-var=omp_get_schedule(&kind,&modifier);

    n = atoi(argv[1]);
    if (n>200)
        n=200;
    chunk = atoi(argv[2]);

    for (i=0; i<n; i++)
        a[i]=i;

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)
    schedule(dynamic,chunk)
    for (i=0; i<n; i++) {
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
            omp_get_thread_num(), i, a[i], suma);

        //Sólo nos piden para un thread por lo que ponemos para el thread 0
        if(omp_get_thread_num() == 0)
        {
            omp_get_schedule(&kind,&modifier);
            printf("Dentro de 'parallel':\n");
            printf("dyn-var: %d\n", omp_get_dynamic());
            printf("nthreads-var: %d\n", omp_get_max_threads());
            printf("thread-limit-var: %d\n", omp_get_thread_limit());
            printf("run-sched-var: %d, %d\n", kind, modifier);
        }
    }
    printf("Fuera de 'parallel for' suma=%d\n", suma);
    omp_get_schedule(&kind,&modifier);
    printf("Fuera de 'parallel':\n");
}
```

```

printf("dyn-var: %d\n", omp_get_dynamic());
printf("nthreads-var: %d\n", omp_get_max_threads());
printf("thread-limit-var: %d\n", omp_get_thread_limit());
printf("run-sched-var: %d, %d\n", kind, modifier);
}

```

CAPTURAS DE PANTALLA:

```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp -O2 scheduled-clause
Modificado.c -o mod
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./mod 4 2
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 10
thread-limit-var: 8
run-sched-var: 2, 1
thread 0 suma a[1]=1 suma=1
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 10
thread-limit-var: 8
run-sched-var: 2, 1
thread 6 suma a[2]=2 suma=2
thread 6 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=5
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 10
thread-limit-var: 8
run-sched-var: 2, 1
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ export OMP_THREAD_LIMIT=4
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ export OMP_SCHEDULE="dynamic"
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./mod 4 2
thread 2 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=2
thread 3 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=5
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 10
thread-limit-var: 4
run-sched-var: 2, 1
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ export OMP_NUM_THREADS=8
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ export OMP_SCHEDULE="static,4"
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./mod 4 2
thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 2 suma a[2]=2 suma=2
thread 2 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=5
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 8
thread-limit-var: 4
run-sched-var: 1, 4
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$

```

RESPUESTA:

Aunque se cambie el valor de las variables de entorno (Tanto dentro como fuera de la región paralela), la suma no cambia.

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char **argv)
{
    int i, n=16, chunk, a[n], suma=0;
    int modifier;
    omp_sched_t kind;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    //dyn-var = omp_get_dynamic();
    //nthreads-var = omp_get_max_threads();
    //thread-limit-var=omp_get_thread_limit();
    //run-sched-var=omp_get_schedule(&kind,&modifier);

    n = atoi(argv[1]);
    if (n>200)
        n=200;
    chunk = atoi(argv[2]);

    for (i=0; i<n; i++)
        a[i]=i;

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)
    schedule(dynamic,chunk)
    for (i=0; i<n; i++) {
        suma = suma + a[i];
        //printf(" thread %d suma a[%d]=%d suma=%d \n",
    omp_get_thread_num(),i,a[i],suma);

    //Sólo nos piden para un thread por lo que ponemos para el thread 0
    if(omp_get_thread_num() == 0)
    {
        omp_get_schedule(&kind,&modifier);
        printf("Dentro de 'parallel':\n");
        printf("dyn-var: %d\n", omp_get_dynamic());
        printf("nthreads-var: %d\n", omp_get_max_threads());
        printf("thread-limit-var: %d\n", omp_get_thread_limit());
        printf("run-sched-var: %d, %d\n", kind, modifier);
        printf(" NUM_THREADS: %d, NUM_PROCS: %d, IN_PARALLEL:%d \n",
    omp_get_num_threads() ,omp_get_num_procs(), omp_in_parallel());
    }
}
```



```

printf("Fuera de 'parallel for' suma=%d\n", suma);
omp_get_schedule(&kind, &modifier);
printf("Fuera de 'parallel':\n");
printf("dyn-var: %d\n", omp_get_dynamic());
printf("nthreads-var: %d\n", omp_get_max_threads());
printf("thread-limit-var: %d\n", omp_get_thread_limit());
printf("run-sched-var: %d, %d\n", kind, modifier);
printf(" NUM_THREADS: %d, NUM_PROCS: %d, IN_PARALLEL: %d \n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
}

```

CAPTURAS DE PANTALLA:

```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp -O2
dificado4.c -o mod4
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./mod4 4 2
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: 2, 1
 NUM_THREADS: 2, NUM_PROCS: 8, IN_PARALLEL:1
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: 2, 1
 NUM_THREADS: 2, NUM_PROCS: 8, IN_PARALLEL:1
Fuera de 'parallel for' suma=5
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: 2, 1
 NUM_THREADS: 1, NUM_PROCS: 8, IN_PARALLEL:0
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./mod4 3 3
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: 2, 1
 NUM_THREADS: 2, NUM_PROCS: 8, IN_PARALLEL:1
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: 2, 1
 NUM_THREADS: 2, NUM_PROCS: 8, IN_PARALLEL:1
Fuera de 'parallel for' suma=3
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: 2, 1
 NUM_THREADS: 1, NUM_PROCS: 8, IN_PARALLEL:0

```

RESPUESTA: Como solos nos piden el valor de las funciones comentamos la iteraciones. El valor de `omp_get_num_threads()` y el de `omp_in_parallel()` cambia cuando estamos fuera de la región parallel mientras que `omp_get_num_procs()` se mantiene.

5. Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char **argv)
{
    int i, n=200, chunk, a[n], suma=0;
    int dyn, nthreads, modifier;
    omp_sched_t kind;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    n = atoi(argv[1]);
    if (n>200)
        n=200;
    chunk = atoi(argv[2]);

    for (i=0; i<n; i++)
        a[i]=i;

    omp_get_schedule(&kind, &modifier);
    printf("Antes de modificar:\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_num_threads());
    printf("run-sched-var: %d, %d\n", kind, modifier);

    omp_set_dynamic(1);
    omp_set_num_threads(1);
    omp_set_schedule(3, 1);

    omp_get_schedule(&kind, &modifier);
    printf("Despues de modificar:\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_num_threads());
    printf("run-sched-var: %d, %d\n", kind, modifier);

    #pragma omp parallel for firstprivate(suma) \
    lastprivate(suma) schedule(dynamic, chunk)
    for (i=0; i<n; i++) {
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
```

```

        omp_get_thread_num(), i, a[i], suma);
    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
}

```

CAPTURAS DE PANTALLA:

```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp -O2
scheduled-clausedModificado5.c -o mod5
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./mod5 16 2
Antes de modificar:
dyn-var: 0
nthreads-var: 1
run-sched-var: 2, 1
Despues de modificar:
dyn-var: 1
nthreads-var: 1
run-sched-var: 3, 1
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[4]=4 suma=5
thread 0 suma a[5]=5 suma=10
thread 0 suma a[6]=6 suma=16
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
thread 1 suma a[8]=8 suma=13
thread 1 suma a[9]=9 suma=22
thread 1 suma a[10]=10 suma=32
thread 1 suma a[11]=11 suma=43
thread 1 suma a[12]=12 suma=55
thread 0 suma a[7]=7 suma=23
thread 0 suma a[14]=14 suma=37
thread 0 suma a[15]=15 suma=52
thread 1 suma a[13]=13 suma=68
Fuera de 'parallel for' suma=52
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$

```

RESPUESTA:

Para modificar las variables de control dyn-var, nthreads-var y run-sched-var usaremos:

```

-omp_set_dynamic(1);
-omp_set_num_threads(1);
-omp_set_schedule(3,1);

```

Después de modificar las variables de control se observa que se ha cambiado su valor

Resto de ejercicios

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

CÓDIGO FUENTE: pmtv-secuencial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char **argv){
    int i, j;
    int N;
    int suma=0;

    int *vector;
    int **matriz;
    int *resultado;

    if (argc < 2){
        printf("Falta el número de componentes\n");
        return(-1);
    }

    N = atoi(argv[1]);

    //Reservamos memoria para los vectores y la matriz
    vector = (int *)malloc(N * sizeof(int));
    resultado = (int *)malloc(N * sizeof(int));
    matriz = (int **)malloc(N * sizeof(int *));

    for (i=0; i<N; i++){
        matriz[i] = (int *)malloc(N * sizeof(int));
    }

    //Inicializamos matriz y vector
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            if (i>j)
                matriz[i][j] = 0;
            else
                matriz[i][j] = j*i;
        }
        vector[i] = j;
    }

    //Multiplicamos la matriz por el vector
    for (i=0; i<N; i++){
        suma=0;
        for (j=0; j<N; j++){
            suma+=(matriz[i][j]*vector[i]);
        }
        resultado[i] = suma;
    }
}
```

```

        printf ("Componente(0,0): %d\n", resultado[0]);
        printf ("Componente (%d,%d): %d\n", N,N, resultado[N-1]);

    for(i=0; i<N; i++)
        free(matriz[i]);
    free(matriz);
    free(resultado);
    free(vector);

    return 0;
}

```

**CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp -O2
pmtv-secuencial.c -o secuencial -lrt
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial 10
Tiempo: 0.000001
Componente(0,0): 0
Componente (9,9): 810
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial 100
Tiempo: 0.000022
Componente(0,0): 0
Componente (99,99): 980100
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial 1000
Tiempo: 0.001017
Componente(0,0): 0
Componente (999,999): 998001000
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$

```

```

albduranlopez@albduranlopez: ~/Dropbox/AC/Tercera Entrega AC/BP2_DuranLopezAlbert
albduranlopez@albduranlopez:~/Dropbox/AC/Tercera Entrega AC/BP2_DuranLopezAlbert
oJesus/codigo$ gcc pmtv-secuencial.c -fopenmp -O2 -lrt -o new
albduranlopez@albduranlopez:~/Dropbox/AC/Tercera Entrega AC/BP2_DuranLopezAlbert
oJesus/codigo$ ./new 100
Tiempo(seg.):0.000016 / Tamaño:100 / V2[0]=328350.000000 V2[99]=818400.0000
00
albduranlopez@albduranlopez:~/Dropbox/AC/Tercera Entrega AC/BP2_DuranLopezAlbert
oJesus/codigo$ ./new 1000
Tiempo(seg.):0.003449 / Tamaño:1000 / V2[0]=332833500.000000 V2[999]=8318340
00.000000
albduranlopez@albduranlopez:~/Dropbox/AC/Tercera Entrega AC/BP2_DuranLopezAlbert
oJesus/codigo$

```

Adjuntamos una segunda captura para comparar el orden de complejidad respecto al código de la práctica anterior y observamos que 'pmtv-secuencial.c' tiene un mejor tiempo a pesar de que tengan la misma eficiencia

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en atcgrid los tiempos de ejecución del código paralelo (usando, como siempre, `-O2` al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto para la alternativa. Use un tamaño de vector `N` múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para `chunk` con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los `chunks`? (c) Con la asignación `dynamic` y `guided`, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

RESPUESTA:

Añadimos `schedule` (runtime)

Planificación y tamaño de bloque determinado por la variable de entorno `OMP_SCHEDULE`

a) En OpenMP, para `dynamic`, si el `chunk` no está especificado, su valor por defecto es 1.

Para `static`, el valor por defecto no está especificado. Las iteraciones se dividen entre los threads.

Para `guided`, su valor por defecto es 1, que define el tamaño mínimo del bloque.

<https://computing.llnl.gov/tutorials/openMP/>

b) Como en `static` se asignan bloques fijos de iteraciones de tamaño “`chunk`” a cada thread, cada uno de ellos hará un trabajo equivalente a $n^{\circ}\text{fila} * \text{chunk}$.

c) `Dynamic` realizará las operaciones más rápido que `guided` ya que conforme las hebras van acabando su trabajo, se asignan automáticamente bloques de trabajo

CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

int main(int argc, char **argv){
    int i, j;
    int N;
    int suma=0;

    int *vector;
    int **matriz;
    int *resultado;

    if (argc < 2){
        printf("Falta el número de componentes\n");
```

```

        return(-1);
    }

    N = atoi(argv[1]);

    //Reservamos memoria para los vectores y la matriz
    vector = (int *)malloc(N * sizeof(int));
    resultado = (int *)malloc(N * sizeof(int));
    matriz = (int **)malloc(N * sizeof(int *));

    for (i=0; i<N; i++){
        matriz[i] = (int *)malloc(N * sizeof(int));
    }

    //Inicializamos matriz y vector
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            if (i>j)
                matriz[i][j] = 0;
            else
                matriz[i][j] = j*i;
        }
        vector[i] = j;
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    //Multiplicamos la matriz por el vector
    #pragma omp parallel for private(j) schedule(runtime)
    for (i=0; i<N; i++){
        suma=0;
        for (j=0; j<N; j++){
            suma+=(matriz[i][j]*vector[j]);
        }
        resultado[i] = suma;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double)
    ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    //Resultados
    printf("Tiempo: %f\n", ncgt);
    printf ("Componente(0,0): %d\n",resultado[0]);
    printf ("Componente (%d,%d): %d\n",N-1,N-1,resultado[N-1]);

    for(i=0; i<N; i++)
        free(matriz[i]);
    free(matriz);
    free(resultado);
    free(vector);

    return 0;
}

```

DESCOMPOSICIÓN DE DOMINIO:

Tenemos la matriz M, el vector V y el resultado R.

A cada $M[1][i]$ se le asigna un $V[i]$, (se reparte y se recorre cada fila de la matriz) y el thread calcula un valor del vector final.

Repetir este proceso para $M[2][i]$, $M[3][i]$, ..., $M[n][i]$

M11	M12	M13	V1	R1

M21	M22	M23	V2	R2

M31	M32	M33	V3	R3

CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
E1estudiante8@atcgrid:~
[E1estudiante8@atcgrid ~]$ qsub 'pmtv-OpenMP_atcgrid.sh' -q ac
57826.atcgrid
[E1estudiante8@atcgrid ~]$ ls
archivo                pmv-OpenMP-a.c          STDIN.e57819  STDIN.o57810
ej7_atcgrid.e57826      pmv-OpenMP-b.c          STDIN.e57822  STDIN.o57813
ej7_atcgrid.o57826      pmv-OpenMP-reduction.c  STDIN.e57823  STDIN.o57819
```

```
albduranlopez@albduranlopez: ~
sftp> get ej7_atcgrid.e57826
Fetching /home/E1estudiante8/ej7_atcgrid.e57826 to ej7_atcgrid.e57826
sftp> get ej7_atcgrid.o57826
Fetching /home/E1estudiante8/ej7_atcgrid.o57826 to ej7_atcgrid.o57826
/home/E1estudiante8/ej7_atcgrid.o57826      100% 1359  1.3KB/s  00:00
sftp>
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash

#PBS -N ej7_atcgrid
#PBS -q ac
echo "Id$PBS_O_WORKDIR usuario de trabajo: $PBS_O_LOGNAME"
echo "Id$PBS_O_WORKDIR de trabajo: $PBS_JOBID"
echo "Nombre del trabajo dado por el usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo: "
cat $PBS_NODEFILE

export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
```



```

$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

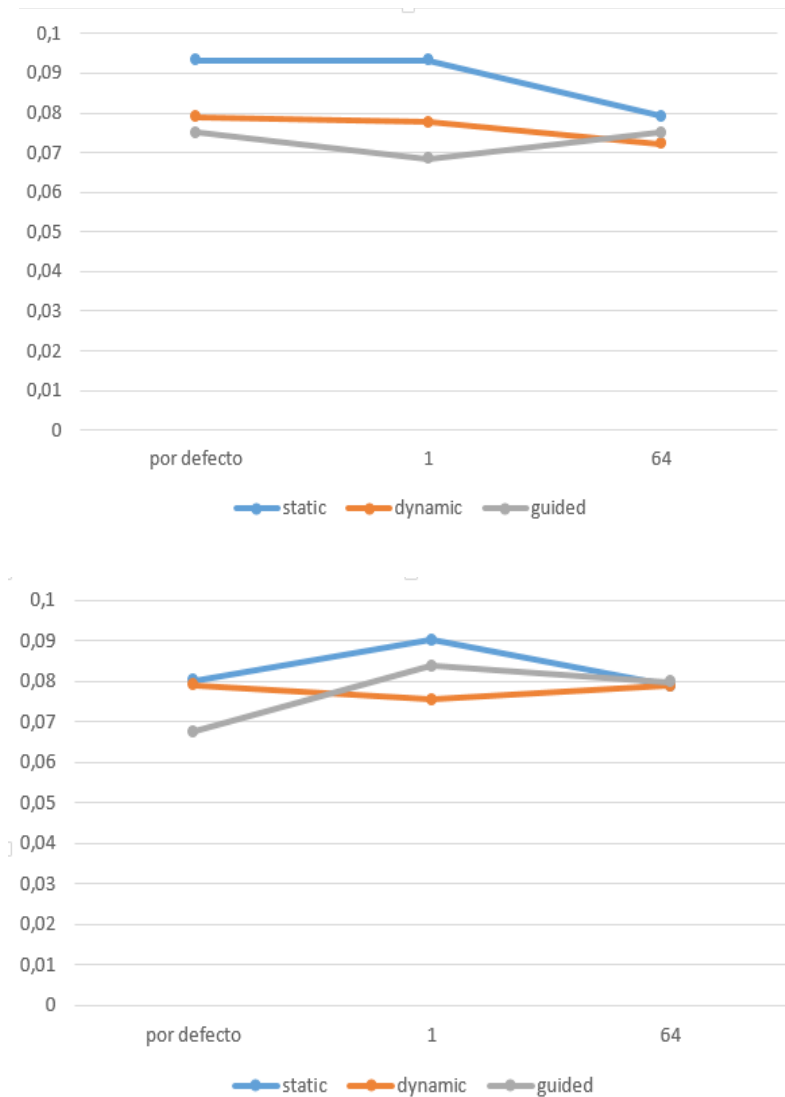
export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

```

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño $N= 15360$, 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.093330	0.078977	0.075097
1	0.093290	0.077643	0.068472
64	0.079064	0.072263	0.075108
Chunk	Static	Dynamic	Guided
por defecto	0.080154	0.078993	0.067534
1	0.090181	0.075529	0.083763
64	0.078833	0.078960	0.079844



8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \cdot C; A(i, j) = \sum_{k=0}^{N-1} B(i, k) \cdot C(k, j), i, j = 0, \dots, N-1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char **argv){
    int i, j, k;
    int N;
    int suma=0;
```

```

int **matrizB;
int **matrizC;
int **matrizA;

if (argc < 2){
    printf("Falta el número de componentes\n");
    return(1);
}

N = atoi(argv[1]);

//Reservamos memoria
matrizB = (int **)malloc(N * sizeof(int*));
matrizC = (int **)malloc(N * sizeof(int*));
matrizA = (int **)malloc(N * sizeof(int*));

for (i=0; i<N; i++){
    matrizB[i] = (int *)malloc(N * sizeof(int));
    matrizC[i] = (int *)malloc(N * sizeof(int));
    matrizA[i] = (int *)malloc(N * sizeof(int));
}

//Inicializamos las matrices
for (i=0; i<N; i++){
    for (j=0; j<N; j++){
        matrizB[i][j] = j+i;
        matrizC[i][j] = j*i;
    }
}

struct timespec cgt1,cgt2; double ncgt;

clock_gettime(CLOCK_REALTIME,&cgt1);

//Multiplicación de matrices
for (i=0; i<N; i++){
    for(j=0; j<N; j++){
        suma = 0;
        for (k=0; k<N; k++){
            suma += (matrizB[i]
[k]*matrizC[k][j]);
        }
        matrizA[i][j]=suma;
    }
}

clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double)
((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

//Resultados
printf("Tiempo: %f\n", ncgt);
printf ("Componente(0,0): %d\n",matrizA[0][0]);
printf ("Componente(%d,%d): %d\n",N-1, N-1, matrizA[N-1][N-1]);

//Liberamos memoria
for(i=0; i<N; i++){
    free(matrizA[i]);
    free(matrizB[i]);
    free(matrizC[i]);
}

```

```

        free(matrizA);
    free(matrizB);
    free(matrizC);

    return 0;
}

```

CAPTURAS DE PANTALLA: (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp -O2 pmm-secuencial.c -o secuencial2 -lrt
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial2 10
Tiempo: 0.000002
Componente(0,0): 0
Componente(9,9): 6210
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial2 100
Tiempo: 0.002809
Componente(0,0): 0
Componente(99,99): 81021600
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial2 1000
Tiempo: 3.045868
Componente(0,0): 0
Componente(999,999): 2073477872
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$

```

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2, Lección 5/Tema 2).

10. DESCOMPOSICIÓN DE DOMINIO:

Dada la matriz M, se reparten y recorren las filas.

M11	M12	M13

M21	M22	M23

M31	M32	M33

Cada thread recorre las filas y también las columnas por lo que tanto los elementos de la fila i como los elementos de la columna i son recorridos por cada thread

CÓDIGO FUENTE: pmm-OpenMP.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

int main(int argc, char **argv){
    int i, j, k;
    int N;
    int suma=0;

    int **matrizB;

```

```

int **matrizC;
int **matrizA;

if (argc < 2){
    printf("Falta el número de componentes\n");
    return(1);
}

N = atoi(argv[1]);

//Reservamos memoria
matrizB = (int **)malloc(N * sizeof(int*));
matrizC = (int **)malloc(N * sizeof(int*));
matrizA = (int **)malloc(N * sizeof(int*));

for (i=0; i<N; i++){
    matrizB[i] = (int *)malloc(N * sizeof(int));
    matrizC[i] = (int *)malloc(N * sizeof(int));
    matrizA[i] = (int *)malloc(N * sizeof(int));
}

//Inicializamos las matrices
#pragma omp parallel for private(j)
for (i=0; i<N; i++){
    for (j=0; j<N; j++){
        matrizB[i][j] = j+i;
        matrizC[i][j] = j*i;
    }
}

struct timespec cgt1,cgt2; double ncgt;

clock_gettime(CLOCK_REALTIME,&cgt1);

//Multiplicación de matrices
#pragma omp parallel for private(k,j)
for (i=0; i<N; i++){
    for(j=0; j<N; j++){
        suma = 0;
        for (k=0; k<N; k++){
            suma += (matrizB[i]
[k]*matrizC[k][j]);
        }
        matrizA[i][j]=suma;
    }
}

clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double)
((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

//Resultados
printf("Tiempo: %f\n", ncgt);
printf ("Componente(0,0): %d\n",matrizA[0][0]);
printf ("Componente(%d,%d): %d\n",N-1, N-1, matrizA[N-1][N-1]);

//Liberamos memoria
for(i=0; i<N; i++){
    free(matrizA[i]);
    free(matrizB[i]);
    free(matrizC[i]);
}

```

```

    }

    free(matrizA);
    free(matrizB);
    free(matrizC);

    return 0;
}

```

CAPTURAS DE PANTALLA: (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

Como podemos observar, los tiempos obtenidos en el código con código en paralelo son menores, como era de esperar.

```

albduranlopez@albduranlopez: ~/Escritorio/Cuarta Entrega AC/código
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp -O2 pmm-secuencial.c -o secuencial2 -lrt
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial2 10
Tiempo: 0.000002
Componente(0,0): 0
Componente(9,9): 6210
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial2 100
Tiempo: 0.002809
Componente(0,0): 0
Componente(99,99): 81021600
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./secuencial2 1000
Tiempo: 3.045868
Componente(0,0): 0
Componente(999,999): 2073477872
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ gcc -fopenmp -O2 pmm-OpenMP.c -o pmmOpenMP -lrt
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./pmmOpenMP 10
Tiempo: 0.000011
Componente(0,0): 0
Componente(9,9): 6210
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./pmmOpenMP 100
Tiempo: 0.001762
Componente(0,0): 0
Componente(99,99): 81021600
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$ ./pmmOpenMP 1000
Tiempo: 1.628502
Componente(0,0): 0
Componente(999,999): 2073477872
albduranlopez@albduranlopez:~/Escritorio/Cuarta Entrega AC/código$

```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

ESTUDIO DE ESCALABILIDAD EN ATCGRID:

SCRIPT: pmm-OpenMP_atcgrid.sh

```

#!/bin/bash

#PBS -N pmm-OpenMP
#PBS -q ac

echo "Id$PBS_O_WORKDIR usuario de trabajo: $PBS_O_LOGNAME"
echo "Id$PBS_O_WORKDIR de trabajo: $PBS_JOBID"
echo "Nombre del trabajo dado por el usuario: $PBS_JOBNAME"

```

```

echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo: "
cat $PBS_NODEFILE

echo "Paralelo 2 threads"
export OMP_NUM_THREADS=2

$PBS_O_WORKDIR/pmm-OpenMP 100
$PBS_O_WORKDIR/pmm-OpenMP 500
$PBS_O_WORKDIR/pmm-OpenMP 1000
$PBS_O_WORKDIR/pmm-OpenMP 1500

echo "Paralelo 4 threads"
export OMP_NUM_THREADS=4

$PBS_O_WORKDIR/pmm-OpenMP 100
$PBS_O_WORKDIR/pmm-OpenMP 500
$PBS_O_WORKDIR/pmm-OpenMP 1000
$PBS_O_WORKDIR/pmm-OpenMP 1500

echo "Paralelo 6 threads"
export OMP_NUM_THREADS=6

$PBS_O_WORKDIR/pmm-OpenMP 100
$PBS_O_WORKDIR/pmm-OpenMP 500
$PBS_O_WORKDIR/pmm-OpenMP 1000
$PBS_O_WORKDIR/pmm-OpenMP 1500

```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pcllocal.sh

```

#!/bin/bash

echo "Paralelo 2 threads"
export OMP_NUM_THREADS=2

./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500

echo "Paralelo 4 threads"
export OMP_NUM_THREADS=4

./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500

echo "Paralelo 6 threads"
export OMP_NUM_THREADS=6

./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500

```

Pc – local

Tamaño	2 threads	4 threads	6 threads
100	0.000459	0.000314	0.000397
500	0.071865	0.040992	0.054873
1000	1.589798	0.860007	1.820245
1500	14.143167	7.097974	8.302739

```

albduranlopez@albduranlopez: ~
albduranlopez@albduranlopez:~$ ./pmm-OpenMP.sh
Paralelo 2 threads
Tiempo: 0.000459
Componente(0,0): 0
Componente(99,99): 81021600
Tiempo: 0.071865
Componente(0,0): 0
Componente(499,499): 252600448
Tiempo: 1.589798
Componente(0,0): 0
Componente(999,999): 2073477872
Tiempo: 14.143167
Componente(0,0): 0
Componente(1499,1499): 1811923920
Paralelo 4 threads
Tiempo: 0.000314
Componente(0,0): 0
Componente(99,99): 81021600
Tiempo: 0.040992
Componente(0,0): 0
Componente(499,499): 252600448
Tiempo: 0.860007
Componente(0,0): 0
Componente(999,999): 2073477872
Tiempo: 7.097974
Componente(0,0): 0
Componente(1499,1499): 1811923920
Paralelo 6 threads
Tiempo: 0.000397
Componente(0,0): 0
Componente(99,99): 81021600
Tiempo: 0.056925
Componente(0,0): 0
Componente(499,499): 252600448
Tiempo: 1.820245
Componente(0,0): 0
Componente(999,999): 2073477872
Tiempo: 8.302739
Componente(0,0): 0
Componente(1499,1499): 1811923920
albduranlopez@albduranlopez:~$ ^C
albduranlopez@albduranlopez:~$

```


Atcgrid

Tamaño	2 threads	4 threads	6 threads
100	0.001656	0.000932	0.001209
500	0.184808	0.096483	0.096614
1000	5.507932	2.793094	1.847310
1500	19.900514	10.046269	6.766402

